Database PerformanceTuning for Developers

Luca Canali, IT-DM

Database Developers’ Workshop

July 8th, 2008

Outline

1. Performance tuning goal(s)
   - Best practices, worst practices and guidelines
2. ‘Know your Oracle’ for better performance and productivity
3. Performance issues analysis

1. Oracle Performance Rules

- Oracle performance
  - – Complex because the engine is multi-purpose
  - – Many advices on the web, but mostly only *partially correct* (aka wrong)

- Andrew Holdsworth’s two simple performance rules:
  - – The key to good performance
    - **You run good execution plans**
    - **There are no serialization points**
  - – Without these all bets are off!

Best Practices

- Best practices are very dangerous
  - Too many parameters in the system to give generic rules
  - Be wary of them
- Standards and guidelines
  - Help the administration, avoid typical pitfalls
  - Overall highly recommended
- Worst practices
  - Much more useful and correct than best practices
  - A current 'trend' at Oracle conferences

Worst Practices

Connection management:

- Short-lived and frequent connections to Oracle are performance and scalability killers
  - Guideline: avoid using Oracle as ’DB connection + short query + disconnect’,
    - Often the case with PHP for example
  - More generally, Oracle likes to work with connection pools / application servers / multi tier applications
  - The database weekly report shows which accounts have large number of logons

Worst Practices

Bind variables

- For OLTP workloads, not using bind variables puts a limit on the application scalability
  - The database weekly report shows statements which could profit from using bind variables
  - For LCG applications using bind variables is often a good idea
  - Beware: there are cases were bind variables can trigger problems or are just not appropriate
    - i.e. ‘use bind variables’ is a typical best practice and should not be blindly followed as such

Worst Practices

Development cycle

- Developing in production is a recipe for disaster
  - Having said that, we still receive about 1 request every 3 months for recovery of accidentally droppped data
  - We also see from time to time, changes in production that cause performance problems
- Guideline for application owners: if you find yourself applying ‘emergency’ changes to prod frequently, please review your development cycle

Guideline

- Development Lifecycle
  - Development DB
  - Pre-production tests (integration DB)
  - Performance and concurrency tests (test DB)
  - Production

- In other words:
  - We promote extensive tests
  - Tests done in a production-like environment
  - We don’t want users to ‘develop in production’

Worst Practices

CERN IT Department

Locks, deadlocks, and serialization points

- Serialization can bring the application to a halt, i.e. servers are wasting CPU cycles
  - Long transactions can be a problem
  - Not indexing foreign keys is a typical mistake
  - Updates and deletes need to be treated with care
    - Do not try concurrent operations on given rows
    - Better move the operation to a ‘daemon’

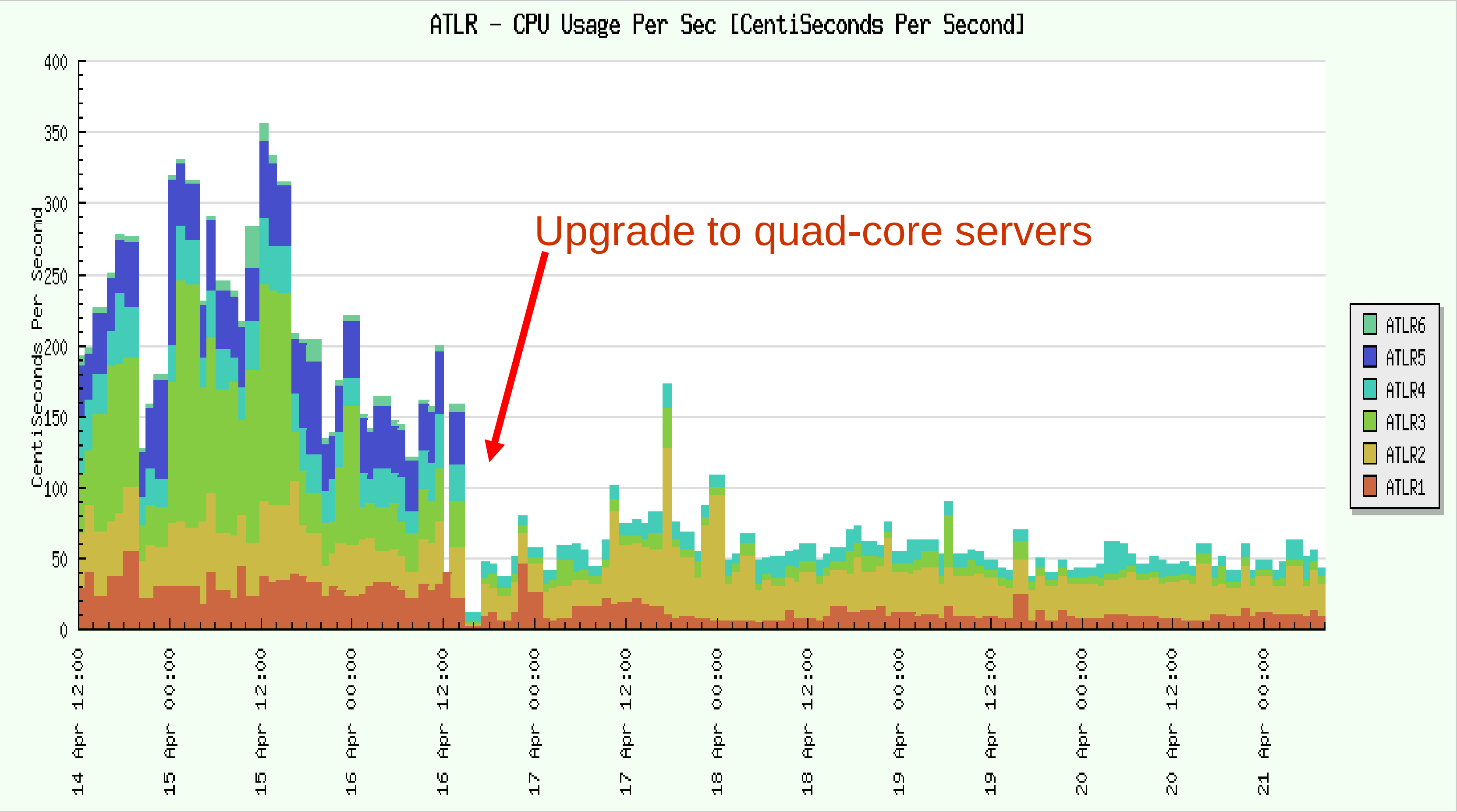
Worst Practices

Indexes: on which attributes? how many indexes?

- Indexes should be carefully chosen at design phase as opposed to being added 'reactively'
  - Unique indexes where the data model needs them
  - Foreign keys without an index are often source of concurrency and performance problems
  - Beware of indexing non-selective columns
  - Composite indexes are often a good idea (the column order is important)
  - Over-indexing is a bad idea for DML performance
  - For special cases, know the various index types available in Oracle

Worst Practice

- Hardware upgrades are not a magic wand
  - Can provide ‘a boost’, but only every few years
  - query tuning can be much more cost-effective



Upgrade to quad-core servers

Guideline: build performance at design time

- Decisions at design time influence performance characteristics
  - Example: indexes speed up reads but make writes slower
  - Several similar trade-off exists
    - see talk on DB design
- Guidelines:
  - Prototype
  - Avoid worst practices
  - Start to do ‘stress testing’ as soon as possible

2. Know your Oracle

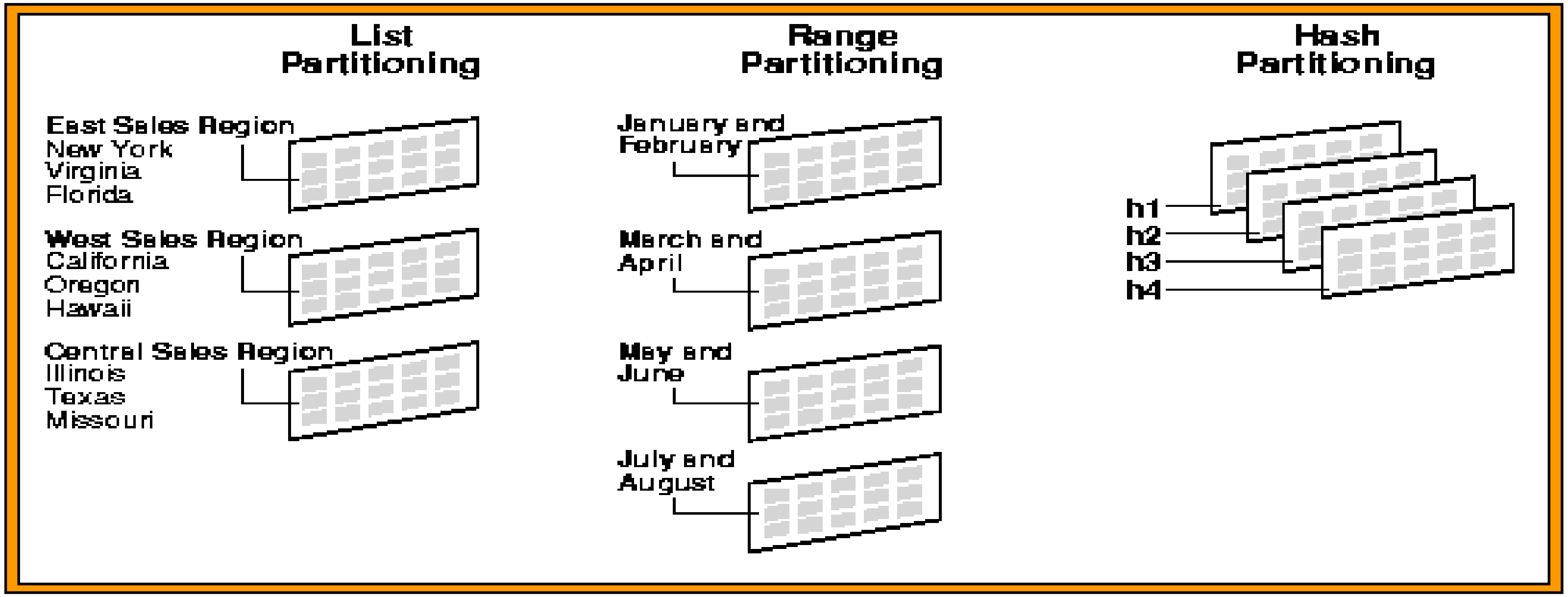
- The more features you know the better you can leverage the potential of Oracle
  - Unfortunately the manual is quite ‘thick’..
    - Certification can help
    - Oracle books (see APRESS series)
    - Blogs (see for example http://jonathanlewis.wordpress.com)
- Guideline: avoid being ‘database independent’, when possible.

Oracle Partitioning

- Tables and indices can be decomposed into smaller and more manageable pieces called *partitions*
  - *Manageability:* data management operations at partition level
    - parallel backup, parallel data loading on independent partitions
  - *Query performance:* partition pruning
    - queries restricted only to the relevant partitions of the table
  - Partitioning is (*almost*) *transparent to users and applications*
    - tables/indices logically unchanged even if physically partitioned!

Oracle Partitioning

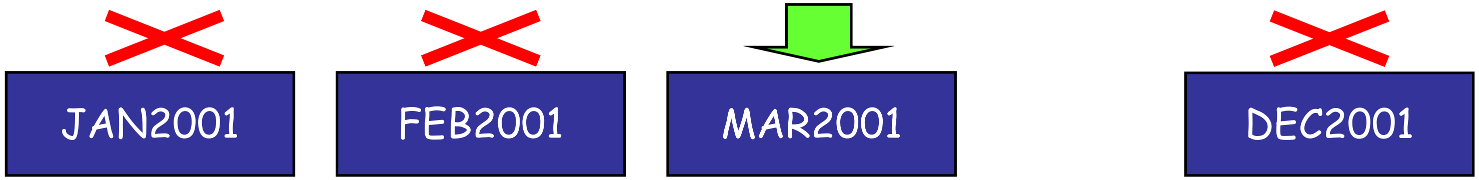
- Several options available



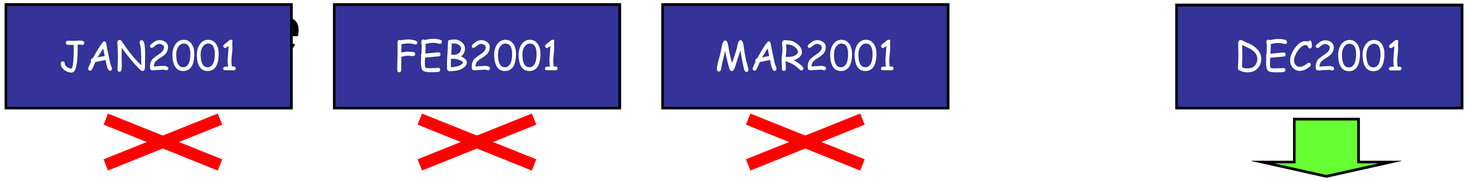
Oracle Partitioning

Loading data into a table partitioned by date range

```
INSERT INTO sales ( …, sale_date, … )
  VALUES ( …, TO_DATE('3-MARCH-2001','dd-mon-yyyy'), … );
```



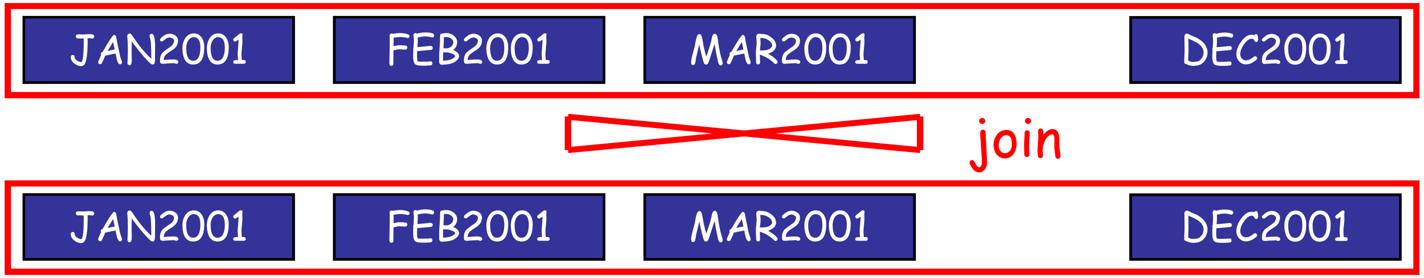
Querying data from a table partitioned by date



```
SELECT … FROM sales
  WHERE sales_date = TO_DATE ('14-DEC-2001','dd-mon-yyyy');
```

Oracle Partitioning

- Without partitioning: global join (query time ~ N x N)



- With partitioning: local joins (query time ~ N)

Oracle Partitioning

- Indexes for partitioned tables can be partitioned too
  - *Local indices: defined within the scope of a partition*

    ```
    CREATE INDEX i_sale_date ON sales
                 (sale_date) LOCAL
    ```

  - In contrast to *global indexes*: defined on the table as a whole

Partitioning Myth

- Example of ‘best practice’, that should be rejected when imposed as a ‘general rule’
  - ‘when a table has several millions of records it needs to be partitioned for performance
- Example: If you access the table with PK only, partitioning may not be useful
  - Example, Compass schema has IOTs > 10^9 rows
- Applications need to be partition-aware
  - In most case of practical use/complexity

Index organized tables (IOT)

- *If a table is most often accessed via a PK*, it may be useful to build the table itself like a B*-tree index!
  - In contrast to standard “heap” tables

- Advantages and disadvantages:
  - Faster queries (no need to look up the real table)
  - Reduced size (no separate index, efficient compression)
  - *But performance may degrade if access is not via the PK*

- IOT syntax (LHCb Bookkeeping example)

```
CREATE TABLE joboptions (
    job_id, name, recipient, value,
CONSTRAINT pk_joboptions PRIMARY KEY (job_id)
) ORGANIZATION INDEX;
```

Composite indexes

- Index over multiple columns in a table
- When to use?
  - When WHERE clause uses more than one column
  - To increase selectivity joining columns of low selectivity
- How to create?
  - Columns with higher selectivity first
  - Columns that can appear isolated in the WHERE clause first
  - Study execution plan of the queries in case of doubt
- FTS example:

```
SELECT max(jobid) FROM t_job
  WHERE channel_name = :b1
  group by vo_name;

CREATE INDEX job_report ON t_job(channel_name, vo_name,
   job_id);
```

- See also Giuseppe’s presentation

Function-based indexes

- Indexes created after applying function to column
  - They speed up queries that evaluate those functions to select data
  - Use only if other options are not possible
  - Typical example, if customers are stored as “ROSS”, “Ross”, “ross” (design problem!):

```
CREATE INDEX customer_name_index
  ON sales (UPPER(customer_name));
```

  - Special case (trick): index on low-cardinality column

```
CREATE INDEX criticality_iscritical ON criticality
   ( CASE WHEN is_critical = ‘Y' THEN ‘Y'
   ELSE NULL
   END);
```

Advanced SQL, analytic functions

- Analytic functions ‘compute an aggregate value based on a group of rows (aka window)’
  - They differ from aggregate functions (‘group by’) in that they return multiple rows for each group
  - Typical functions who can be used: AVG, COUNT, MAX, MIN, SUM,DENSE_RANK, RANK, LEAD
- Example:

```
SELECT t_job.job_id id, ..
     DENSE_RANK() OVER ( ORDER BY t_job.priority DESC,
     SYS_EXTRACT_UTC(t_job.submit_time) ) TopJob
FROM t job..
```

Analytic functions, Top-N example

```
SQL> select ename, sal,
row_number() over (order by sal desc) rn
from emp
order by sal desc;

ENAME SAL RN
----- ---- --
KING  5000  1
FORD  3000  2
SCOTT 3000  3
...

 (example from ask tom)
```

Advanced Queuing

- Oracle comes with a API for queue handling
  - Publish and subscribe
  - Propagation
- Example: It’s used by Streams replication to manage LCR (database changes):
  - Enqueue the change
  - Propagation to remote instance
  - Dequeueing of the LCR by multiple apply slaves.
- Guideline: no need to recreate a queue mechanism with Oracle tables and custom code

Sequences

- Sequences are the most scalable way to generate sequential and unique numbers
  - Typically used in a Primary key
  - Ex: select mysequence.nexval from mysequence;

- The use of a table instead of sequences is lethal for scalability, especially in RAC
  - EX: update my_seq_table set my_seq_table+1

- In RAC use cache and noordering
  - EX: **`create sequence mysequence start with 1 maxvalue 10000 cache 50 noorder;`**

RAC and current block waits

- Applications that write concurrently into a given table (or set of tables)
  - Typically don’t scale well in RAC
  - The reason is that an Oracle (data) block can be acquired for writing (current mode) by 1 instance at a time
  - Shipping current blocks is a potential scalability bottleneck
- Possible solutions
  - Move the application to single node
  - Do writes/updates in asynch mode,
    - delegate DML operations to a daemon
    - or implement a queue-based system
    - or partition data make the application partitoin and instance aware

SQL Hints – advanced query tuning

- Tuning of execution plans as generated by the Oracle optimizer (CBO)
  - CBO does not always generate the best plan
  - Plan stability can be an issue
  - See also Giuseppe's and Andrea's talks

- Examples of hints:
  - ALL_ROWS optimizes for best throughput
  - FIRST_ROWS optimizes for best response time to get the first rows…
  - FULL chooses a full table scan
    - It will disable the use of any index on the table
  - INDEX chooses an index scan for the table
  - INDEX_JOIN will merge the scans on several (single-) column indexes

3. Analysis of performance issues

- You have avoided the known worst practices and followed guidelines, but still have a performance problem.

- What to do next?

Guideline: workflow

- Gather data
- Build a model
- Build and implement change
- Check results against the baseline and the goal
- Repeat

Gather data

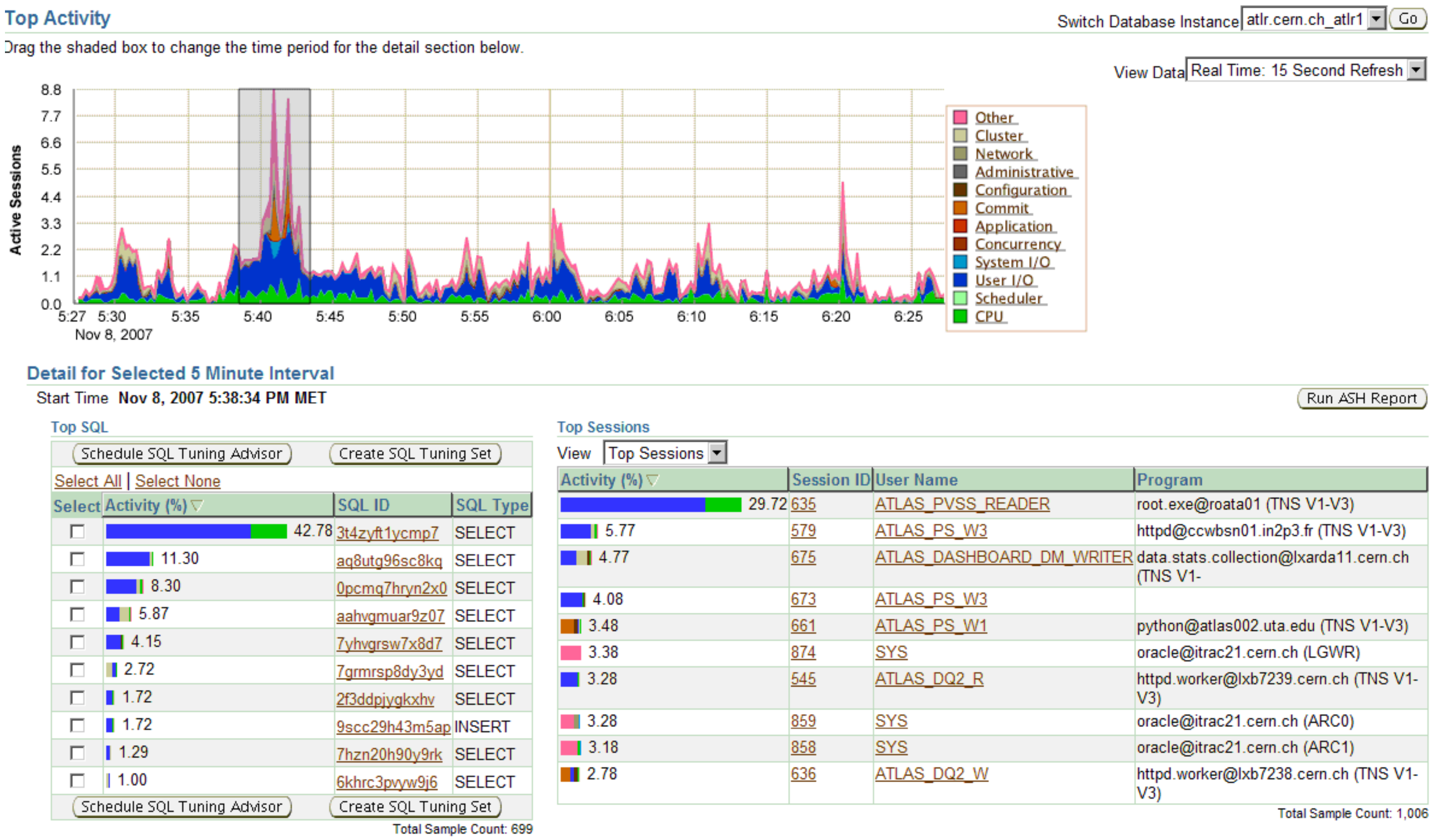
- Response time analysis
  - Measure as much as you can
  - Instrument code so that you can see where time is spent
  - Goal, identify where time is spent, for example slow sql

Gather data

Database level

- With the use of sql trace (aka 10046 event) the DBA can ‘dump the SQL execution’
- It shows which SQL is slow and in which part of the execution
- Wait events information is available
- EM and other monitoring tools
    - AWR reports (system metric and wait events ‘deltas’)

Gather data



Top Activity

Drag the shaded box to change the time period for the detail section below.

Detail for Selected 5 Minute Interval

Start Time **Nov 8, 2007 5:38:34 PM MET**

Top SQL

Select	Activity (%) ▽	SQL ID	SQL Type
□	42.78	3t4zyft1ycmp7	SELECT
□	11.30	aq8utg96sc8kq	SELECT
□	8.30	0pcmq7hryn2x0	SELECT
□	5.87	aahvgmuar9z07	SELECT
□	4.15	7yhvgrsw7x8d7	SELECT
□	2.72	7grmrsp8dy3yd	SELECT
□	1.72	2f3ddpjygkxhv	SELECT
□	1.72	9scc29h43m5ap	INSERT
□	1.29	7hzn20h90y9rk	SELECT
□	1.00	6khrc3pvyw9j6	SELECT

Total Sample Count: 699

Top Sessions

Activity (%) ▽	Session ID	User Name	Program
29.72	635	ATLAS_PVSS_READER	root.exe@roata01 (TNS V1-V3)
5.77	579	ATLAS_PS_W3	httpd@ccwbsn01.in2p3.fr (TNS V1-V3)
4.77	675	ATLAS_DASHBOARD_DM_WRITER	data.stats.collection@lxarda11.cern.ch (TNS V1-
4.08	673	ATLAS_PS_W3	
3.48	661	ATLAS_PS_W1	python@atlas002.uta.edu (TNS V1-V3)
3.38	874	SYS	oracle@itrac21.cern.ch (LGWR)
3.28	545	ATLAS_DQ2_R	httpd.worker@lxb7239.cern.ch (TNS V1-V3)
3.28	859	SYS	oracle@itrac21.cern.ch (ARC0)
3.18	858	SYS	oracle@itrac21.cern.ch (ARC1)
2.78	636	ATLAS_DQ2_W	httpd.worker@lxb7238.cern.ch (TNS V1-V3)

Total Sample Count: 1,006

Gather data 2

- Execution plan
  - Study the execution plan of the slow queries
  - Identify bad plans
  - Typical case: a full scan on a table which should be accessed by index

Example

- EXPLAIN PLAN
  - SQL command that allows to find out what is the execution plan *before* the SQL statement runs

```
SQL> EXPLAIN PLAN FOR
  SELECT file_state FROM lcg_fts_prod.t_file
  WHERE file_id = :B1;

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
--------------------------------------------------------
| Id  | Operation                     | Name             |
--------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |
|   1 |  TABLE ACCESS BY INDEX ROWID| T_FILE           |
|*  2 |   INDEX UNIQUE SCAN           | FILE_FILE_ID_PK  |
--------------------------------------------------------

Predicate Information (identified by operation id):
----------------------------------------------------

   2 - access("FILE_ID"=TO_NUMBER(:B1))
```

- Use a tool (e.g. Benthic Golden - Ctrl-P)

Example

- from SQL*Plus (the plan used by the actual statement execution)

```
SQL> set autotrace traceonly
SQL> var :b1 number;
SQL> exec :b1 := 3423
SQL> SELECT file_state FROM lcg_fts_prod.t_file
  WHERE file_id = :B1;
----------------------------------------------------------------------------------
| Id  | Operation                     | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |     1 |    11 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T_FILE           |     1 |    11 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN           | FILE_FILE_ID_PK |     1 |       |     0   (0)| 00:00:01 |
----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("FILE_ID“=TO_NUMBER(:B1))

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          1  consistent gets
          1  physical reads
          0  redo size
        279  bytes sent via SQL*Net to client
        385  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

Build a Model

- Response time analysis
  - From the data make a model of where is the time spent
  - Is it in the DB or application?
  - If it’s in the DB, on which statements?

Build and Implement Change

- From the model
  - Make a informed decision on what to change and apply the change
  - One change at a time helps to analyze the results
- Typical actions
  - Query tuning
    - Creation or drop of an index
    - Restructuring of the query
  - Sudden change in performance
    - Bind variable peeking issue, may require hints to stabilize the query
    - One-off issue (application restart)

Evaluate results

- Performing one change at a time in analyzing the impact of the change
- The tuning cycle should continue until the goals are met
  - Example: meeting a given response time measured at the application level is a good goal
  - A baseline (measured acceptable performance) is another defined goal.

Comment on perf analysis

- The exercise described in the previous slides is best done in collaboration
  - DBA and application developers
- The integration and test environments are available
  - And are the best places to perform this type of tuning
  - Critical applications need stress testing with ‘real-world data’

Conclusions

- Application performance benefits from
  - A good design
  - Building performance (stress) testing into the development cycle from an early stage
  - Experience and information: guidelines, standards, worst practices to avoid, Oracle (new features)
- The IT-DM DBA team is available to provide consultancy

Acknowledgments

- This presentation contains the work of the IT-DM DBA team: Dawid, Eva, Jacek, Luca, Maria, Miguel.
- Many thanks to the experiment DBAs and developers who have worked with us on application tuning